CHAPITRE 5 : RECHERCHE DE MOTIFS

1. Présentation du problème

1.1 Recherche d'un motif dans un texte

Problème. Rechercher toutes les occurrences d'un motif x dans un texte t. On généralisera ensuite à la recherche de plusieurs motifs $x_1, \ldots x_n$, puis à la recherche d'une expression régulière r (comme on a vu dans le TP sur lex).

Ce problème est omniprésent en informatique : éditeur de texte, moteur de recherche, analyse de séquences biologiques, imagerie informatique, composition musicale automatique, etc.

Il suscite encore de nombreuses recherches, à la fois du point de vue pratique (pour optimiser les méthodes), et du point de vue théorique.

1.2 Principe général

Méthode de base (Morris & Pratt) :

• On construit l'AFD qui reconnaît $\Sigma^* x$

• On analyse le texte t avec l'AFD : chaque fois qu'on passe par un état final, c'est qu'on a trouvé une occurrence de x

Coût de la recherche = coût de la construction + coût de la lecture

Il existe de nombreuses variantes de cette méthode.

1.3 Tableau des différentes méthodes

Les méthodes de recherche de motif peuvent être classées en deux familles :

1- celles où le motif x est fixe (comme ci-dessus),

2- celles où c'est le texte t qui est fixe.

Dans le cas 1, il existe deux fameux algorithmes :

- Knuth, Morris & Pratt : c'est une variante de la méthode de base avec construction de l'AFD pour $\Sigma^* x$. Dans ce cas, le texte t est comparé à x en commençant par le début de x (état initial de l'AFD). Les échecs se traduisent par le passage par certaines transitions de l'AFD qui reviennent à décaler x dans le texte (voir plus loin).

- Boyer & Moore : la comparaison entre x et t se fait en commençant par la fin de x (de droite à gauche), et s'il y a échec, on décale x dans le texte avec un saut judicieusement choisi.

Dans le cas 2, la méthode générale est basée sur l'AFD reconnaissant les suffixes de t. Cet AFD ne doit pas nécessairement être complet, mais il doit être accessible et co-accessible (tout état doit pouvoir être atteint de l'état initial, et de tout état, on doit pouvoir atteindre un état final). On lit x dans l'AFD : si on peut le lire intégralement, c'est qu'on a lu un préfixe d'un suffixe de t, donc une occurrence de x dans t.

	Lecture gauche-droite	Lecture droite-gauche
Motif x fixe	Morris & Pratt = AFD de $\Sigma^* x$	AFD des suffixes du miroir $x\sim$
	Knuth, Morris & Pratt (variante de la fonction d'échec)	Boyer & Moore (variante)
Texte t fixe	AFD des suffixes de t	

2. Recherche d'un motif fixe par lecture gauche-droite

2.1 Méthode de l'AFD de $\Sigma^* x$

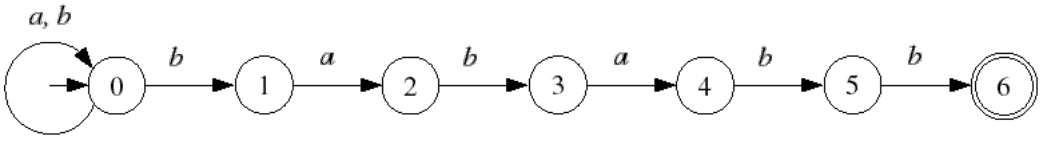
L'écorché d'un mot x de longueur n est l'AFD obtenu avec $n + 1$ états, et n flèches correspondant aux lettres successives du mot. Dans cet automate, on voit clairement que les états ont un rôle de « mémoire » : chaque état mémorise la position dans le mot x.

Pour reconnaître $\Sigma^* x$, il suffit de rajouter une boucle sur l'état initial avec toutes les lettres de l'alphabet. On obtient un AFN.

On a vu que la déterminisation d'un AFN pouvait être très coûteuse (2^n états pour un AFN ayant n états). Il est remarquable que dans le cas de $\Sigma^* x$, le coût de la déterminisation soit faible.

Proposition. *L'AFD obtenu en déterminisant l'écorché d'un mot x de longueur, augmenté d'une boucle sur l'état initial avec toutes les lettres, n a au plus n états.*

Exemple : $x = bababb$



Après déterminisation, on obtient l'AFD suivant. La correspondance entre les états de l'AFN et ceux de l'AFD est (0 est inchangé) :

1 = {0, 1}

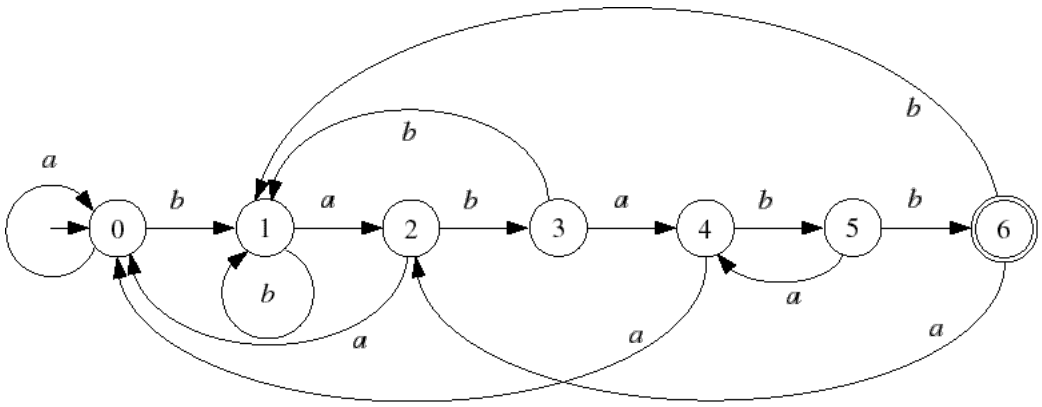
2 = {0, 2}

3 = {0, 1, 3}

4 = {0, 2, 4}

5 = {0, 1, 3, 5}

6 = {0, 1, 6}



Méthode de recherche : le texte *t* est comparé à *x* en commençant par <u>le début de *x*</u> (état initial de l'AFD). Les échecs se traduisent par le passage par certaines transitions de l'AFD qui reviennent à décaler *x* dans le texte

Exemple : recherche de *x* = *bababb*, dans *t* = *babbabaababababbaababababab*

b a b b a b a a b a b a b b a a b a b a b a b a b

1 2 3 1

retour à l'état 1 : on se décale pour recommencer avec le préfixe *b* de *x*

c'est-à-dire que <u>*b* est le plus long préfixe de *bab* qui soit aussi suffixe</u>

b a b b a b a a b a b a b b a a b a b a b a b a b

1 2 3 1 2 3 4 0

retour à l'état 0 : on se décale pour recommencer au début de *x*

b a b b a b a a b a b a b b a a b a b a b a b a b

1 2 3 1 2 3 4 0 1 2 3 4 5 <u>6</u> 2

<u>état 6 final</u> : *x* = *bababb* est trouvé

retour à l'état 2 : on se décale pour recommencer avec le préfixe *ba* de *x*

b a b b a b a a b a b a b b a a b a b a b a b a b

1 2 3 1 2 3 4 0 1 2 3 4 5 6 2 0

retour à l'état 0 : on se décale pour recommencer au début de *x*

b a b b a b a a b a b a b b a a b a b a b a b a b

1 2 3 1 2 3 4 0 1 2 3 4 5 6 0 0 1 2 3 4 5 4

retour à l'état 4 : on se décale pour recommencer avec le préfixe *baba* de *x*

b a b b a b a a b a b a b b a a b a b a b a b a b

1 2 3 1 2 3 4 0 1 2 3 4 5 6 0 0 1 2 3 4 5 4 5 4

retour à l'état 4 : on se décale pour recommencer avec le préfixe *baba* de *x*, mais il ne reste plus assez de lettres dans *t*, donc stop

2.2 Fonction d'échec : algorithme de Morris & Pratt

On peut calculer directement l'AFD pour $\Sigma^* x$, sans passer par la déterminisation de l'AFN, grâce à une fonction d'échec.

Si *p* est un état de l'écorché de *x* (autre que 0) permettant de lire *w*, on définit *f*(*p*) de la manière suivante :

f(*p*) = état d'arrivée du plus long suffixe propre de *w* qui est aussi préfixe de *w* (donc de *x*)

Pour calculer facilement *f*(*p*), il faut voir si on peut le déduire de *f*(*p*-1). Il se trouve que ça marche.

i = *f*(*p*-1)

• si la lettre *x*(*i*+1) = *x*(*p*)

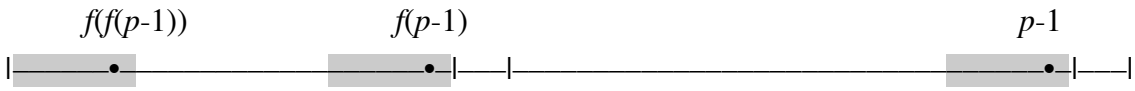
alors *x*(1)…*x*(*i*+1) est à la fois préfixe et suffixe,

donc on peut prolonger le suffixe précédent : $f(p) = i + 1 = f(p\text{-}1) + 1$

• si la lettre est différente, il faut essayer un suffixe plus court, lequel ?

i = ?

=> ça doit être <u>un suffixe du suffixe précédent</u>



$i = f(f(p\text{-}1))$
et on recommence la comparaison entre $x(i+1)$ et $x(p)$

<u>calcul de la fonction d'échec f</u>

$f(0) \leftarrow -1$
$i \leftarrow -1$
pour $p = 1$ **à** n
    tant que $i \geq 0$ **et** $x(i+1) \neq x(p)$
        $i \leftarrow f(i)$
    $i \leftarrow i+1$
    $f(p) \leftarrow i$

Exemple : Calcul de f pour le motif $x = bababb$

$f(0) = -1$, $i = -1$
$p = 1$ $i = -1$, on incrémente $i = 0$, et pose $f(1) = 0$

$p = 2$ $i = 0$, $x(1) = b \neq x(2) = a$, on recule $i = f(0) = -1$
on incrémente $i = 0$, et on pose $f(2) = 0$

$p = 3$ $i = 0$, $x(1) = b = x(3)$, on incrémente $i = 1$, et on pose $f(3) = 1$

$p = 4$ $i = 1$, $x(2) = a = x(4)$, on incrémente $i = 2$, et on pose $f(4) = 2$

$p = 5$ $i = 2$, $x(3) = b = x(5)$, on incrémente $i = 3$, et on pose $f(5) = 3$

$p = 6$ $i = 3$, $x(4) = a \neq x(6) = b$, on recule $i = f(3) = 1$
$x(2) = a \neq x(6) = b$, on recule $i = f(1) = 0$
on incrémente $i = 1$, et on pose $f(6) = 1$

D'où la fonction d'échec :

État p	0	1	2	3	4	5	6
$f(p)$		0	0	1	2	3	1

Exemple :

• pour $p = 2$, le mot lu est *ba*, il n'a pas de suffixe propre qui soit préfixe, donc $f(2) = 0$

• pour $p = 3$, le mot lu est *bab*, le plus long suffixe propre qui soit préfixe est *b*, donc $f(3) = 1$

• pour $p = 4$, le mot lu est *baba*, le plus long suffixe propre qui soit préfixe est *ba*, donc $f(4) = 2$

La fonction d'échec s'utilise de la manière suivante : quand dans un état donné p on ne peut pas lire la lettre courante du texte t, on réessaie à partir de $f(p)$. Si w est le mot lu en arrivant à l'état p, alors par définition de la fonction d'échec, celui lu en arrivant à l'état $f(p)$ est un suffixe de w qui est aussi préfixe de w, donc du motif x. Donc c'est un début possible pour une occurrence de x.

Exemple : reprenons le motif $x = bababb$.

On le recherche dans $t = bababbabb$

b a b a b a *b b*

b a b a b

1 2 3 4 5 échec dans la lecture : *t* contient *a*, alors qu'il faut un deuxième *b*

3 on reprend à l'état $3 = f(5)$, ce qui correspond au préfixe *bab*

b a b a b a b b

3 4 5 6

Avec la fonction d'échec, on peut aussi reconstruire directement l'AFD reconnaissant $\Sigma^* x$. Dans l'exemple ci-dessus, après échec en 5, on reprend en $3 = f(5)$ pour pouvoir lire *a* (de l'état 3 à 4) après avoir lu le préfixe *bab*. Dans l'AFD de $\Sigma^* x$, cela correspond exactement à la transition de 5 vers 4 étiquetée par *a* (voir dessin de l'AFD).

On complète la fonction de transition de la manière suivante :

pour toute lettre *a* telle que $\delta(p, a)$ non défini, on pose

• $\delta(p, a) = \delta(f(p), a)$ si $p \neq 0$,

• $\delta(0, a) = 0$.

On vérifie facilement qu'en rajoutant ces transitions, on obtient exactement l'AFD vu plus haut.

L'algorithme de Morris & Pratt ne construit pas explicitement l'AFD de $\Sigma^* x$. Il se contente de lire le texte t en avançant dans l'AFD reconnaissant x (l'écorché de x), et en effectuant les retours arrière définis par la fonction d'échec. Il fonctionne de la manière suivante, en supposant données

• la fonction de transition $\delta(p, a)$ de l'écorché de x,

• la fonction d'échec $f(p)$.

$p \leftarrow 0$ (état initial)

tant que t non vide

avancer dans t, $a \leftarrow 1^{ère}$ lettre de t

tant que $p \neq 0$ **et** $\delta(p, a)$ non défini

$p \leftarrow f(p)$

si $\delta(p, a)$ défini **alors** $p \leftarrow \delta(p, a)$

si $p \in F$ **alors** signaler occurrence de x dans t

Une variante de cet algorithme est l'algorithme de Knuth, Morris & Pratt. Il consiste à optimiser la fonction d'échec.

3. Recherche d'un motif fixe par lecture droite-gauche

3.1 Méthode de l'AFD des suffixes du motif miroir

Le miroir $x\sim$ d'un mot x est le mot lu à l'envers (en commençant par la fin).

Méthode :

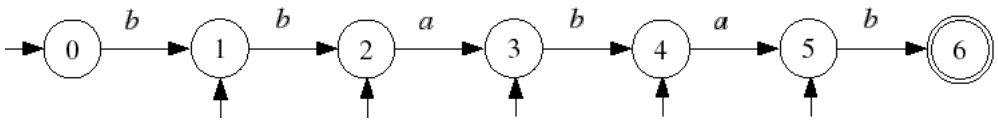
• On construit l'AFD qui reconnaît les suffixes du mot miroir $x\sim$

• On analyse le texte t avec l'AFD par une fenêtre de longueur $|x|$, dans laquelle on lit de droite à gauche : en cas d'échec, on revient au dernier état final de l'AFD rencontré. Il correspond à un suffixe s de $x\sim$. Donc de gauche à droite, $s\sim$ est un préfixe de x. On se décale dans t pour commencer une nouvelle lecture à partir de $s\sim$. Ces sauts accélèrent considérablement la recherche.

Exemple : motif $x = bababb$

AFN pour les suffixes de $x\sim = bbabab$

On part de l'écorché de $x\sim$, et on rend initiaux tous les états sauf le dernier.



Après déterminisation, on obtient l'AFD suivant. La correspondance entre les états de l'AFN et ceux de l'AFD est :

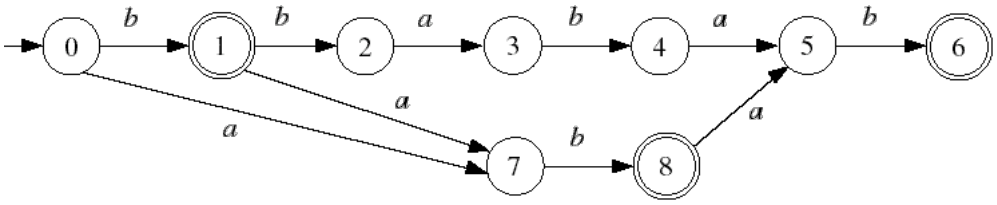
0 = {0, 1, 2, 3, 4, 5}

1 = {1, 2, 4, 6}

2, 3, 4, 5, 6 sont identiques

7 = {3 , 5}

8 = {4, 6}



Recherche de *x* = *bababb*, dans *t* = *babbabaababababbaababababab*

b a b b a b a a b a b a b b a a b a b a b a b a b

8 7 1 0

état 8 (final) : on ne peut lire *b* dans l'AFD, *bab* est le plus long sufixe de *x*~ trouvé, on se décale pour recommencer avec le préfixe *bab* de *x*

b a b b a b a a b a b a b b a a b a b a b a b a b

7 1 0

état 7 : on ne peut lire *a* dans l'AFD, *ba* est le mot lu (non suffixe), on revient à l'état 1 (final) : *b* le plus long suffixe de *x*~ trouvé, on se décale pour recommencer avec le préfixe *b* de *x*

b a b b a b a a b a b a b b a a b a b a b a b a b

6 5 4 3 2 1 0

x = *bababb* est trouvé

on revient à l'état final 1 : *b* le plus long suffixe de *x*~ trouvé, on se décale pour recommencer avec le préfixe *b* de *x*

b a b b a b a a b a b a b b a a b a b a b a b a b

5 8 7 1 0

état 5 : on ne peut lire *a* dans l'AFD, *baba* est le mot lu (non suffixe), on revient à l'état 8 (final) : *bab* le plus long suffixe de *x*~ trouvé, on se décale pour recommencer avec le préfixe *bab* de *x*

b a b b a b a a b a b a b b a a b a b a b a b a b

6 5 8 7 0

état 6 (final) : on ne peut lire *a* dans l'AFD, *abab* est le plus long suffixe de *x*~ trouvé, on se décale pour recommencer avec le préfixe *baba* de *x*

b a b b a b a a b a b a b b a a b a b a b a b

6 5 8 7 0

état 6 (final) : on ne peut lire *a* dans l'AFD, *abab* est le plus long suffixe de *x*~ trouvé, on se décale pour recommencer avec le préfixe *baba* de *x*, mais il ne reste plus assez de lettres dans *t*, donc stop

3.2 Variante : algorithme de Boyer & Moore

Dans la méthode précédente avec l'AFD des suffixes du miroir *x*~, en cas d'échec dans la lecture droite-gauche, l'AFD permet de trouver un préfixe du mot qu'on a lu qui est suffixe de *x*~, c'est-à-dire dont le miroir est préfixe de *x*. On se décale pour recommencer la lecture à partir de ce préfixe de *x,* qui peut être un début possible pour une nouvelle occurrence.

Dans l'algorithme de Boyer & Moore, qui est une variante, mais sans construire l'AFD précédent, on ne repart pas d'un préfixe plus court que le mot lu dans la fenêtre droite-gauche (donné par l'AFD), mais on utilise ce mot complet (on note *w* son miroir, dans l'ordre normal gauche-droite) :

1) on numérote les positions dans la fenêtre de 1 à *n* (où *n* est la longueur du motif *x*) dans l'ordre normal gauche-droite,
2) on construit une fonction *f*(*i*) qui donne la position où le suffixe de *x* qu'on a lu *w* = *x*(*i*)…*x*(*n*) réapparaît plus à gauche dans le motif *x*,
3) on se décale dans le texte *t* pour faire coïncider la fenêtre avec cette occurrence de *w*.

Exemple : recherche de *x* = *baacaa*, dans *t* = *aaabaacaa*

On calcule la fonction *f*, en attribuant la valeur 0 aux suffixes qui ne réapparaissent pas plus à gauche dans le mot.

Position	1	2	3	4	5	6
f(*i*)	0	0	0	0	2	5

a a a b a a c a a

5 6 échec de la lecture du *c* attendu au lieu de *b*, suffixe lu = *aa*

on décale la fenêtre pour la faire coïncider avec le *aa* trouvé, en utilisant *f*(5) = 2

a a a b a a c a a

1 2 3 4 5 6 occurrence trouvée

La différence avec l'AFD des suffixes de *x*~ est que le décalage ne garantit pas qu'on repart d'un début de *x*. Avant l'occurrence du suffixe mise en coïncidence avec une portion du motif, il peut y avoir des lettres divergentes qui font échec.

Exemple : Si le texte avait été $t = aaadaacaa$ (avec d à la place de b), le décalage aurait donné un échec.

En fait, dans l'algorithme de Boyer & Moore, on utilise aussi le caractère qui fait l'échec (ici le b trouvé à la place du c) pour améliorer le décalage. Cela ne change pas le calcul ci-dessus dans le premier exemple. Mais dans le deuxième avec d à la place de b, ça aurait évité le décalage donnant un échec.

4. Recherche d'un ensemble fini X de motifs, ou d'une expression régulière r

4.1 Recherche d'un ensemble fini X de motifs

On a un ensemble de motifs $X = \{x_1, \ldots x_k\}$.

Les deux principales méthodes ci-dessus se généralisent à ce cas :

1) Construction de l'AFD pour $\Sigma^* X$
2) Construction de l'AFD pour l'ensemble des suffixes des miroirs $X\sim$

4.2 Recherche d'une expression régulière r

On construit un AFN qui reconnaît $\Sigma^* r$ grâce à l'algorithme de Thompson.

Puis soit on le déterminise en AFD, soit on fait la recherche en simulant l'AFD (sans construire tous les états, mais seulement ceux dont on a besoin au fur et à mesure pour la lecture du texte t).

5. Remarques complémentaires

5.1 Comparaison des deux méthodes : Morris & Pratt, suffixes du miroir

La méthode de l'AFD de $\Sigma^* x$ avec une représentation de l'AFD par une table de transition, a un coût de construction en $O(|x||\Sigma|)$, et un coût pour la lecture de t en $O(|t|)$.

Si l'on ne construit par effectivement l'AFD, mais qu'on construit seulement la fonction d'échec, le coût de la construction est en $O(|x|)$, et celui de l'analyse en $O(|t|\log|\Sigma|)$.

Dans l'exemple traité ci-dessus, la méthode de Morris & Pratt (AFD de $\Sigma^* x$) donnait les mêmes sauts que celle de l'AFD des suffixes de $x\sim$. Mais ce n'est pas vrai dans le cas général. Voici un exemple qui montre les différences entre les deux méthodes.

Exemple : recherche de $x = bbabbabb$, dans $t = abaabbabbabb$

1) méthode de Morris & Pratt

Position	1	2	3	4	5	6	7	8

$f(i)$	0	1	0	1	2	3	4	5

$\boxed{a\ b\ b\ b\ b\ a\ b\ b}\ a\ b\ b$

0 échec dans la lecture : t contient a, alors qu'il faut un b

on se décale d'une lettre

$a\ \boxed{b\ b\ b\ b\ a\ b\ b}\ a\ b\ b$

1 2 échec dans la lecture : t contient b, alors qu'il faut un a

1 on reprend à l'état $1 = f(2)$

$a\ b\ \boxed{b\ b\ b\ a\ b\ b\ a\ b}\ b$

1 2 échec dans la lecture : t contient b, alors qu'il faut un a

1 on reprend à l'état $1 = f(2)$

$a\ b\ b\ \boxed{b\ b\ a\ b\ b\ a\ b\ b}$

1 2 3 4 5 6 7 8 occurrence de x trouvée

Dans cet exemple, la fonction de Morris & Pratt ne permet d'éviter aucun saut : on essaie toutes les positions de la fenêtre jusqu'à l'occurrence de x.

2) méthode de l'AFD de $x\sim$

Dans ce cas, x est identique à son miroir, on a : $x\sim = x = bbabbabb$.

$\boxed{a\ b\ b\ b\ b\ a\ b\ b}\ a\ b\ b$

$b\ b\ a\ b\ b$ échec dans la lecture droite-gauche

Le principe est de repartir du plus long préfixe w de x lu dans la fenêtre droite-gauche (c'est-à-dire tel que $w\sim$ est suffixe de $x\sim$). Ici $w = bbabb$ est le plus long préfixe trouvé, donc on décale la fenêtre pour la faire coïncider avec $bbabb$.

$a\ b\ b\ \boxed{b\ b\ a\ b\ b\ a\ b\ b}$

occurrence de x trouvée

Dans ce cas, on ne fait qu'un seul saut, et le calcul est bien meilleur qu'avec Morris & Pratt.

5.2 Simulation d'un AFD pour $\Sigma^* r$

Voilà un exemple illustrant la simulation dynamique d'un AFD, c'est-à-dire la construction de certains états de l'AFD au fur et à mesure où on en a besoin, sans passer par la construction préalable de tous les états.

• Construction de l'AFN reconnaissant l'expression régulière $\Sigma^* r$ par l'algorithme de Thompson

• Analyse du texte en simulant l'AFD correspondant

Pour P un ensemble d'états de l'AFN, on note

Cloture(P) = les états accessibles à partir de $q \in P$ par ε-transition

Transition(P, a) = clôture de l'union des $\delta(q, a)$ pour $q \in P$

L'analyse de t dans l'AFN se fait en partant de Cloture(I) où I est l'ensemble des états initiaux, et en suivant les flèches indiquées par Transition(P, a).

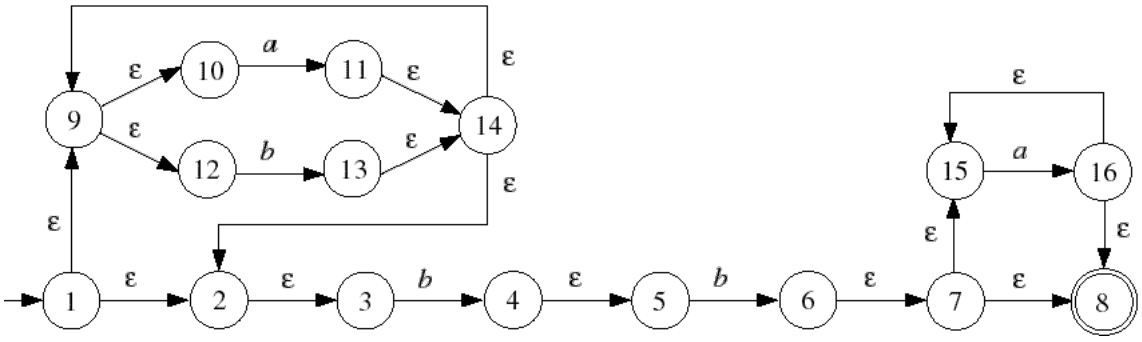
Exemple : recherche du motif $r = bba^*$ dans le texte $t = babbaab$

1) algorithme de Thompson pour l'AFN de $(a + b)^*bba^*$

• pour la concaténation : on sépare les éléments concaténés par des ε-transitions

• pour l'union : on regroupe les éléments de la disjonction par des ε-transitions qui partent d'un même état de départ et se rejoignent sur un même état d'arrivée

• pour l'étoile : on boucle de l'état d'arrivée vers celui de départ par une ε-transition



2) analyse de $t = babbaab$

initialisation : $P = \{1, 2, 3, 9, 10, 12\}$

transition par b : {4, 13}

clôture : {4, 5, 13, 14, 9, 10, 12, 2, 3}

transition par a : {11}

clôture : {11, 14, 9, 10, 12, 2, 3}

transition par *b* : {4, 13}
clôture : {4, 5, 13, 14, 9, 10, 12, 2, 3}

transition par *b* : {4, 6, 13}
clôture : {4, 5, 13, 6, 7, 8, 15, 14, 9, 10, 12, 2, 3}
8 est final, donc une occurrence a été détectée

transition par *a* : {11, 16}
clôture : {11, 14, 9, 10, 12, 2, 3, 16, 8, 15}
une occurrence a été détectée

transition par *a* : {11, 16}
clôture : {11, 14, 9, 10, 12, 2, 3, 16, 8, 15}
une occurrence a été détectée

transition par *b* : {4, 13}
clôture : {4, 5, 13, 14, 9, 10, 12, 2, 3}

Remarque : dans ce cas, la construction de l'AFD par déterminisation n'aurait pas été trop coûteuse, car l'AFN pour bba^* a 3 états, et on est dans le cas où l'AFD pour $\Sigma^* bba^*$ a le même nombre d'états.